目錄

Consul 简介和快速入门

> 来源：vincentmi/consul-guide

翻译自官方文档 .

欢迎进入Consul的入门指南!这个指南是开始使用Consul的起点,通过这个指南了解Consul是什么,他可以解决哪些问题.它与现有软件的比较和怎么开始使用它.如果你对Consul已经有基本的了解,可以阅读 文档 ,它提供更多可用特性的参考.

英文原版

https://www.consul.io/intro/

翻译

工作需要看了下,顺便翻译了。翻译有不当的地方请帮忙指正。

Vincent Mi http://vnzmi.com
miwenshu#gmail.com

Consul是什么

Consul包含多个组件,但是作为一个整体,为你的基础设施提供服务发现和服务配置的工具.他提供以下关键特性:

- 服务发现 Consul的客户端可用提供一个服务,比如 api 或者mysql ,另外一些客户端可用使用Consul去发现一个指定服务的提供者.通过DNS或者HTTP应用程序可用很容易的找到他所依赖的服务.
- 健康检查 Consul客户端可用提供任意数量的健康检查,指定一个服务(比如:webserver是否返回了200 OK 状态码)或者使用本地节点(比如:内存使用是否大于90%). 这个信息可由operator用来监视集群的健康.被服务发现组件用来避免将流量发送到不健康的主机.
- **Key/Value**存储 应用程序可用根据自己的需要使用Consul的层级的Key/Value存储.比如动态配置,功能标记,协调,领袖选举等等,简单的HTTP API让他更易于使用.
- 多数据中心: Consul支持开箱即用的多数据中心.这意味着用户不需要担心需要建立额外的抽象层让业务扩展到多个区域.

Consul面向DevOps和应用开发者友好.是他适合现代的弹性的基础设施.

基础架构

Consul是一个分布式高可用的系统. 这节将包含一些基础,我们忽略掉一些细节这样你可以快速了解Consul是如何工作的.如果要了解更多细节,请参考深入的架构描述.

每个提供服务给Consul的节点都运行了一个Consul agent . 发现服务或者设置和获取 key/value存储的数据不是必须运行agent.这个agent是负责对节点自身和节点上的服务进行健康检查的.

Agent与一个和多个Consul Server 进行交互.Consul Server 用于存放和复制数据.server自行选举一个领袖.虽然Consul可以运行在一台server , 但是建议使用3到5台来避免失败情况下数据的丢失.每个数据中心建议配置一个server集群.

你基础设施中需要发现其他服务的组件可以查询任何一个Consul 的server或者agent.Agent会自动转发请求到server .

每个数据中运行了一个Consul server集群.当一个跨数据中心的服务发现和配置请求创建时.本地Consul Server转发请求到远程的数据中心并返回结果.

Consul与其他软件的比较

https://www.consul.io/intro/vs/

使用Consul

Consul集群的每个节点都必须先安装Consul.安装非常容易,Consul发布为所支持的平台和架构的二进制包.这个指南不包含从源代码编译Consul的内容.

安装Consul

安装Consul,找到适合你系统的包下载他.Consul打包为一个'Zip'文件.

下载后解开压缩包.拷贝Consul到你的PATH路径中,在Unix系统中 `~/bin` 和 `/usr/local/bin` 是通常的安装目录.根据你是想为单个用户安装还是给整个系统安装来选择.在Windows系统中有可以安装到 `%PATH%` 的路径中.

OS X

如果你使用 `homebrew` 作为包管理器,你可以使用命令

```
brew install consul
```

来进行安装.

验证安装

完成安装后,通过打开一个新终端窗口检查 `consul` 安装是否成功.通过执行 `consul` 你应该看到类似下面的输出

```
[root@hdp2 ~]# consul
usage: consul [--version] [--help] <command> [<args>]

Available commands are:
    agent            Runs a Consul agent
    configtest       Validate config file
    event            Fire a new event
    exec             Executes a command on Consul nodes
    force-leave      Forces a member of the cluster to enter the "
left" state
    info             Provides debugging information for operators
    join             Tell Consul agent to join cluster
    keygen           Generates a new encryption key
    keyring          Manages gossip layer encryption keys
    leave            Gracefully leaves the Consul cluster and shut
s down
    lock             Execute a command holding a lock
    maint            Controls node or service maintenance mode
    members          Lists the members of a Consul cluster
    monitor          Stream logs from a Consul agent
    reload           Triggers the agent to reload configuration fi
les
    rtt              Estimates network round trip time between nod
es
    version          Prints the Consul version
    watch            Watch for changes in Consul
```

如果你得到一个`consul not be found`的错误,你的`PATH`可能没有正确设置.请返回检查你的`consul`的安装路径是否包含在`PATH`中.

运行Agent

完成Consul的安装后,必须运行agent. agent可以运行为server或client模式.每个数据中心至少必须拥有一台server . 建议在一个集群中有3或者5个server.部署单一的server,在出现失败时会不可避免的造成数据丢失.

其他的agent运行为client模式.一个client是一个非常轻量级的进程.用于注册服务,运行健康检查和转发对server的查询.agent必须在集群中的每个主机上运行.

查看启动数据中心的细节请查看这里.

启动 Agent

为了更简单,现在我们将启动Consul agent的开发模式.这个模式快速和简单的启动一个单节点的Consul.这个模式不能用于生产环境,因为他不持久化任何状态.

```
[root@hdp2 ~]# consul agent -dev
==> Starting Consul agent...
==> Starting Consul agent RPC...
==> Consul agent running!
         Node name: 'hdp2'
        Datacenter: 'dc1'
            Server: true (bootstrap: false)
       Client Addr: 127.0.0.1 (HTTP: 8500, HTTPS: -1, DNS: 8600,
 RPC: 8400)
      Cluster Addr: 10.0.0.52 (LAN: 8301, WAN: 8302)
    Gossip encrypt: false, RPC-TLS: false, TLS-Incoming: false
             Atlas: <disabled>

==> Log data will now stream in as it occurs:

    2016/08/17 15:20:41 [INFO] serf: EventMemberJoin: hdp2 10.0.
0.52
    2016/08/17 15:20:41 [INFO] serf: EventMemberJoin: hdp2.dc1 1
0.0.0.52
    2016/08/17 15:20:41 [INFO] raft: Node at 10.0.0.52:8300 [Fol
lower] entering Follower state
    2016/08/17 15:20:41 [INFO] consul: adding LAN server hdp2 (A
ddr: 10.0.0.52:8300) (DC: dc1)
    2016/08/17 15:20:41 [INFO] consul: adding WAN server hdp2.dc
1 (Addr: 10.0.0.52:8300) (DC: dc1)
    2016/08/17 15:20:41 [ERR] agent: failed to sync remote state
: No cluster leader
    2016/08/17 15:20:42 [WARN] raft: Heartbeat timeout reached,
starting election
    2016/08/17 15:20:42 [INFO] raft: Node at 10.0.0.52:8300 [Can
didate] entering Candidate state
    2016/08/17 15:20:42 [DEBUG] raft: Votes needed: 1
    2016/08/17 15:20:42 [DEBUG] raft: Vote granted from 10.0.0.5
2:8300. Tally: 1
    2016/08/17 15:20:42 [INFO] raft: Election won. Tally: 1
    2016/08/17 15:20:42 [INFO] raft: Node at 10.0.0.52:8300 [Lea
der] entering Leader state
    2016/08/17 15:20:42 [INFO] raft: Disabling EnableSingleNode
(bootstrap)
    2016/08/17 15:20:42 [DEBUG] raft: Node 10.0.0.52:8300 update
d peer set (2): [10.0.0.52:8300]
    2016/08/17 15:20:42 [INFO] consul: cluster leadership acquir
ed
    2016/08/17 15:20:42 [DEBUG] consul: reset tombstone GC to in
dex 2
    2016/08/17 15:20:42 [INFO] consul: member 'hdp2' joined, mar
king health alive
    2016/08/17 15:20:42 [INFO] consul: New leader elected: hdp2
    2016/08/17 15:20:43 [INFO] agent: Synced service 'consul'
```

如你所见,Consul Agent 启动并输出了一些日志数据.从这些日志中你可以看到,我们的agent运行在server模式并且声明作为一个集群的领袖.额外的本地镀锌被标记为一个健康的成员.

> OS X用户注意: Consul 使用你的主机hostname作为默认的节点名字.如果你的主机名包含时间,到这个节点的DNS查询将不会工作.为了避免这个情况,使用 `-node` 参数来明确的设置node名.

集群成员

新开一个终端窗口运行 `consul members` , 你可以看到Consul集群的成员.下一节我们将讲到加入集群.现在你应该只能看到一个成员,就是你自己:

```
[root@hdp2 ~]# consul members
Node  Address          Status  Type    Build  Protocol  DC
hdp2  10.0.0.52:8301  alive   server  0.6.4  2         dc1
```

这个输出显示我们自己的节点.运行的地址,健康状态,自己在集群中的角色,版本信息.添加 `-detailed` 选项可以查看到额外的信息.

`members` 命令的输出是基于gossip协议是最终一致的.意味着,在任何时候,通过你本地agent看到的结果可能不是准确匹配server的状态.为了查看到一致的信息,使用HTTP API(将自动转发)到Consul Server上去进行查询:

```
[root@hdp2 ~]#  curl localhost:8500/v1/catalog/nodes
[{"Node":"hdp2","Address":"10.0.0.52","TaggedAddresses":{"wan":"
10.0.0.52"},"CreateIndex":3,"ModifyIndex":4}]
```

除了HTTP API ,DNS 接口也可以用来查询节点.注意,你必须确定将你的DNS查询指向Consul agent的DNS服务器,这个默认运行在 `8600` 端口.DNS条目的格式(例如:"Armons-MacBook-Air.node.consul")将在后面讲到.

```
$ dig @127.0.0.1 -p 8600 Armons-MacBook-Air.node.consul
...

;; QUESTION SECTION:
;Armons-MacBook-Air.node.consul.    IN  A

;; ANSWER SECTION:
Armons-MacBook-Air.node.consul. 0 IN    A   172.20.20.11
```

停止Agent

你可以使用Ctrl-C 优雅的关闭Agent. 中断Agent之后你可以看到他离开了集群并关闭.

在退出中,Consul提醒其他集群成员,这个节点离开了.如果你强行杀掉进程.集群的其他成员应该能检测到这个节点失效了.当一个成员离开,他的服务和检测也会从目录中移除.当一个成员失效了,他的健康状况被简单的标记为危险,但是不会从目录中移除.Consul会自动尝试对失效的节点进行重连.允许他从某些网络条件下恢复过来.离开的节点则不会再继续联系.

此外,如果一个agent作为一个服务器,一个优雅的离开是很重要的,可以避免引起潜在的可用性故障影响达成一致性协议.

查看这里了解添加和移除server.

注册服务

在之前的步骤我们运行了第一个agent.看到了集群的成员,查询节点,在这个指南我们将注册我们的第一个服务并查询这些服务.

定义一个服务

可以通过提供服务定义或者调用HTTP API来注册一个服务.服务定义文件是注册服务的最通用的方式.所以我们将在这一步使用这种方式.我们将会建立在前一步我们覆盖的代理配置。

首先,为Consul配置创建一个目录.Consul会载入配置文件夹里的所有配置文件.在Unix系统中通常类似 `/etc/consul.d` (.d 后缀意思是这个路径包含了一组配置文件).

```
$ sudo mkdir /etc/consul.d
```

然后,我们将编写服务定义配置文件.假设我们有一个名叫 `web` 的服务运行在 80端口.另外,我们将给他设置一个标签.这样我们可以使用他作为额外的查询方式:

```
echo '{"service": {"name": "web", "tags": ["rails"], "port": 80}
}' \
    >/etc/consul.d/web.json
```

现在重启agent , 设置配置目录:

```
$ consul agent -dev -config-dir /etc/consul.d
==> Starting Consul agent...
...
    [INFO] agent: Synced service 'web'
...
```

你可能注意到了输出了 "synced" 了 web这个服务.意思是这个agent从配置文件中载入了服务定义,并且成功注册到服务目录.

如果你想注册多个服务,你应该在Consul配置目录创建多个服务定义文件.

查询服务

一旦agent启动并且服务同步了.我们可以通过DNS或者HTTP的API来查询服务.

DNS API

让我们首先使用DNS API来查询.在DNS API中,服务的DNS名字是 `NAME.service.consul`. 虽然是可配置的,但默认的所有DNS名字会都在 `consul` 命名空间下.这个子域告诉Consul,我们在查询服务, `NAME` 则是服务的名称.

对于我们上面注册的Web服务.它的域名是 `web.service.consul` :

```
[root@hdp2 consul.d]# dig @127.0.0.1 -p 8600 web.service.consul

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.47.rc1.el6 <<>> @127.0.0.1 -p
 8600 web.service.consul
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46501
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONA
L: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;web.service.consul.           IN      A

;; ANSWER SECTION:
web.service.consul.     0      IN      A       10.0
.0.52

;; Query time: 0 msec
;; SERVER: 127.0.0.1#8600(127.0.0.1)
;; WHEN: Wed Aug 17 19:07:05 2016
;; MSG SIZE  rcvd: 70
```

如你所见,一个 `A` 记录返回了一个可用的服务所在的节点的IP地址. `` `A `` 记录只能设置为IP地址. 有也可用使用 DNS API 来接收包含 地址和端口的 SRV记录:

```
[root@hdp2 ~]# dig @127.0.0.1 -p 8600 web.service.consul SRV

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.47.rc1.el6 <<>> @127.0.0.1 -p
 8600 web.service.consul SRV
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33415
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONA
L: 1
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;web.service.consul.           IN        SRV

;; ANSWER SECTION:
web.service.consul.     0          IN         SRV        1 1
80 hdp2.node.dc1.consul.

;; ADDITIONAL SECTION:
hdp2.node.dc1.consul.   0          IN         A          10.0
.0.52

;; Query time: 1 msec
;; SERVER: 127.0.0.1#8600(127.0.0.1)
;; WHEN: Thu Aug 18 10:40:48 2016
;; MSG SIZE  rcvd: 130
```

`SRV` 记录告诉我们 `web` 这个服务运行于节点 `hdp2.node.dc1.consul` 的 `80` 端口. DNS额外返回了节点的A记录.

最后,我们也可以用 DNS API 通过标签来过滤服务.基于标签的服务查询格式为 `TAG.NAME.service.consul` . 在下面的例子中,我们请求Consul返回有 `rails` 标签的 `web` 服务.我们成功获取了我们注册为这个标签的服务:

```
[root@hdp2 ~]# dig @127.0.0.1 -p 8600 rails.web.service.consul S
RV

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.47.rc1.el6 <<>> @127.0.0.1 -p
 8600 rails.web.service.consul SRV
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3517
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONA
L: 1
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;rails.web.service.consul.         IN          SRV

;; ANSWER SECTION:
rails.web.service.consul. 0        IN          SRV        1 1 80
hdp2.node.dc1.consul.

;; ADDITIONAL SECTION:
hdp2.node.dc1.consul.     0          IN          A          10.0
.0.52

;; Query time: 1 msec
;; SERVER: 127.0.0.1#8600(127.0.0.1)
;; WHEN: Thu Aug 18 11:26:17 2016
;; MSG SIZE  rcvd: 142
```

HTTP API

除了DNS API之外,HTTP API也可以用来进行服务查询:

```
[root@hdp2 ~]# curl http://localhost:8500/v1/catalog/service/web
[{"Node":"hdp2","Address":"10.0.0.52","ServiceID":"web","Service
Name":"web","ServiceTags":["rails"],"ServiceAddress":"","Service
Port":80,"ServiceEnableTagOverride":false,"CreateIndex":4,"Modif
yIndex":254}]
```

目录API给出所有节点提供的服务.稍后我们会像通常的那样带上健康检查进行查询.就像DNS内部处理的那样.这是只查看健康的实例的查询方法:

```
[root@hdp2 ~]# curl http://localhost:8500/v1/catalog/service/web
?passing
[{"Node":"hdp2","Address":"10.0.0.52","ServiceID":"web","Service
Name":"web","ServiceTags":["rails"],"ServiceAddress":"","Service
Port":80,"ServiceEnableTagOverride":false,"CreateIndex":4,"Modif
yIndex":254}]
```

更新服务

服务定义可以通过配置文件并发送 `SIGHUP` 给agent来进行更新.这样你可以让你在不关闭服务或者保持服务请求可用的情况下进行更新.

另外 HTTP API可以用来动态的添加,移除和修改服务.

建立集群

我们开始了第一个agent并且在agent上注册并查询了服务.这些展示了Consul是如何的易用.但是我们还不知道Consul如何进行扩容成一个可扩展,面向生成环境的服务发现架构.这一章我们将创建我们第一个拥有多个成员的真正的集群.

当一个agent启动时,他开始不知道其他节点的信息,他是一个成员的孤立集群.为了了解其他集群成员这个agent必须加入一个已经存在的集群.要加入一个已经存在的集群,只需要知道一个已经存在的集群成员.通过与这个成员的沟通来发现其他成员,Consul agent可以加入任何agent而不只是出于server模式的agent.

启动Agent

>

> 官方版本教程里使用了Vagrant来启动虚拟机.我已经创建了多个虚拟机
> 因此跳过这部分

我们启动了另外的2台主机,10.0.0.53 ,10.0.0.54 和之前安装的方式一样,将consul拷贝到 `PATH` 目录完成安装.

在之前的示例中,我们使用了 `-dev` 参数来快速的创建一个开发模式的server.然而这并不能充分的在集群环境下使用.现在我们将忽略掉 `-dev` 标签,用我们的集群选项来替换他.

每个集群中的节点都必须要一个唯一的名字.Consul默认会使用机器的hostname.我们可以使用 `-node` 手动覆盖他.

我们也可以使用-bind指定一个绑定的地址让Consul在这个地址上进行监听,这个地址必须可以被其他集群成员访问到.绑定地址不是必须提供,Consul选择第一个私有IP进行监听,不过最好还是指定一个.生产环境的服务器通常有多个网络接口.所以指定一个不会让Consul绑错网络接口.

第一个节点将扮演集群的唯一server,我们使用 `-server` 指定他.

`-bootstrap-expect` 选项提示Consul我们期待加入的server节点的数量.这个选项的作用是启动时推迟日志复制直到我们期望的server都成功加入时.你可以阅读启动指南了解更多.

最后,我们加入 `config-dir` 选项,指定服务和健康检查定义文件存放的路径.

加到一起,命令如下:

```
consul agent -server -bootstrap-expect 1  -data-dir /tmp/consul
-node=hdp2 -bind=10.0.0.52  -config-dir /etc/consul.d
```

现在在另外一个终端,我们将连接第二个节点:

```
ssh hdp3
# 登录第二台机器
```

这一次我们设置绑定的IP地址为第二个节点的IP的地址,并指定节点名称.因为这个节点将不是Consule的server.我们没有打开 `server` 开关.命令如下:

```
consul agent -data-dir /tmp/consul -node=hdp3 -bind=10.0.0.53 -config-dir /etc/consul.d
```

现在,你有运行了两个Consul的agent,一个作为server另一个作为client.这两个agent还互相不知道对方,只是作为独立的单节点集群.为了验证这个你可以在每个agent运行 `consul member` ,只能看到各自自己这一个集群成员.

加入一个集群

现在我们告诉第一个agent来加入第二个agent,在新的终端运行如下命令

```
ssh hdp2
consul join 10.0.0.53
Successfully joined cluster by contacting 1 nodes.
```

>

> 如果出现 `Error joining the cluster: dial tcp 10.0.0.53:8301: getsockopt: no`
>
> 可能是业务防火墙的原因,检查端口 `8301` 是否被允许

你应该可以看到在每个agent的日志输出窗口的一些输出.如果你仔细阅读会发现.他们收到了加入信息,如果你在每个agent运行 `consul members` 你会看到类似下面的内容:

```
[root@hdp2 ~]# consul members
Node  Address          Status  Type    Build  Protocol  DC
hdp2  10.0.0.52:8301   alive   server  0.6.4  2         dc1
hdp3  10.0.0.53:8301   alive   client  0.6.4  2         dc1
```

>

> 记住:为了加入集群,一个Consul的agent只需要了解一个已经存在的集群成员.加入集群后agent会自动交流传递完整的成员信息.

启动时自动加入集群

理想的情况,当一个新的节点在数据中心启动时,他应该自动加入到Consul的集群中,而不需要人为干预.为了达到这个效果你可以使用HashiCorp的Atlas
和 `-atlas-join` 选项.示例如下:

```
consul agent -atlas-join \
  -atlas=ATLAS_USERNAME/infrastructure \
  -atlas-token="YOUR_ATLAS_TOKEN"
```

Atlas的用户名和token可以通过创建Atlas账号获取.这样当新的节点启动后他会自动加入到你的Consul集群,不需要硬编码配置.

另一种选择,你可以通过 `-join` 选项和 `start_join` 配置将其他已知的agent的地址进行硬编码来在启动时加入集群.

查询节点

就像查询服务一样.Consul有一个API用来查询节点自己.你可以通过DNS和HTTP的API来进行.

DNS API中节点名称结构为 `NAME.node.consul` 或者 `NAME.node.DATACENTER.consul` .如果数据中心名字省略,Consul只会查询本地数据中心.

例如 从节点 `hdp2` 我们可以查询节点 `hdp3` 的地址:

```
[root@hdp2 ~]# dig @127.0.0.1 -p 8600 hdp3.node.consul

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.47.rc1.el6 <<>> @127.0.0.1 -p
 8600 hdp3.node.consul
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5351
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONA
L: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;hdp3.node.consul.            IN        A

;; ANSWER SECTION:
hdp3.node.consul.         0         IN        A         10.0
.0.53

;; Query time: 1 msec
;; SERVER: 127.0.0.1#8600(127.0.0.1)
;; WHEN: Thu Aug 18 14:32:02 2016
;; MSG SIZE  rcvd: 66
```

除服务之外查询节点的能力对于系统管理任务非常重要.例如知道节点的SSH登录地址,可以简单的将节点加入到Consul集群并查询他.

离开集群

离开集群,你可以 `Ctrl-C` 优雅的退出,也可以直接Kill掉agent进程.优雅的退出可以让节点转变为离开状态.否则节点将被标记为失败.详细的细节可以查看这里.

健康检查

我们现在看到Consul运行时如此简单.添加节点和服务,查询节点和服务.在这一节.我们将继续添加健康检查到节点和服务.健康检查是服务发现的关键组件.预防使用到不健康的服务.

这一步建立在前一节的Consul集群创建之上.目前你应该有一个包含两个节点的Consul集群.

定义检查

和服务类似,一个检查可以通过检查定义或HTTP API请求来注册.

我们将使用和检查定义来注册检查.和服务类似,因为这是建立检查最常用的方式.

在第二个节点的配置目录建立两个定义文件:

```
vagrant@n2:~$ echo '{"check": {"name": "ping",
  "script": "ping -c1 163.com >/dev/null", "interval": "30s"}}'
\
  >/etc/consul.d/ping.json

vagrant@n2:~$ echo '{"service": {"name": "web", "tags": ["rails"
], "port": 80,
  "check": {"script": "curl localhost >/dev/null 2>&1", "interva
l": "10s"}}}' \
  >/etc/consul.d/web.json
```

第一个定义增加了一个主机级别的检查,名字为 "ping" . 这个检查每30秒执行一次,执行 `ping -c1 163.com` . 在基于脚本的健康检查中,脚本运行在与Consul进程一样的用户下.如果这个命令以非0值退出的话这个节点就会被标记为不健康.这是所有基于脚本的健康检查的约定.

第二个命令定义了名为 `web` 的服务,添加了一个检查.每十分钟通过curl发送一个请求,确定web服务器可以访问.和主机级别的检查一样.如果脚本以非0值退出则标记为不健康.

现在重启第二个agent或者发送 `SIGHUP` 信号,你应该可以看到如下的日志内容输出:

```
==> Reloading configuration...
    2016/08/18 15:29:57 [INFO] agent: Synced service 'web'
    2016/08/18 15:29:57 [INFO] agent: Synced check 'ping'
    2016/08/18 15:29:58 [WARN] agent: Check 'service:web' is now
 critical
```

前几行检查到agent同步了新的定义.最后一行检查到web服务出于危险状态.这是因为我们实际上没有运行一个web服务器.所以 `` `curl `` 的测试会一直失败!

检查健康状态

现在我们加入了一些简单的检查.我们能适应HTTP API来检查他们.首先我们检查有哪些失败的检查.使用这个命令(注意:这个命令可以运行在任何节点)

```
[root@hdp3 consul.d]# curl http://localhost:8500/v1/health/state
/critical
[{"Node":"hdp3","CheckID":"service:web","Name":"Service 'web' ch
eck","Status":"critical","Notes":"","Output":"","ServiceID":"web
","ServiceName":"web","CreateIndex":878,"ModifyIndex":878}]
```

我们可以看到,只有一个检查我们的 `web` 服务在 `critical` 状态

另外,我们可以尝试用DNS查询web服务,Consul将不会返回结果.因为服务不健康.

```
[root@hdp3 consul.d]# dig @127.0.0.1 -p 8600 web.service.consul

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.47.rc1.el6 <<>> @127.0.0.1 -p
 8600 web.service.consul
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 33096
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONA
L: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;web.service.consul.            IN         A

;; AUTHORITY SECTION:
consul.            0           IN          SOA        ns.consul. p
ostmaster.consul. 1471507354 3600 600 86400 0

;; Query time: 8 msec
;; SERVER: 127.0.0.1#8600(127.0.0.1)
;; WHEN: Thu Aug 18 16:02:34 2016
;; MSG SIZE  rcvd: 104
```

下一步

在本章,你学到了如何鉴定的添加健康检查.检查定义可以通过配合文件并发送 `SIGHUP` 到agent进行更新.另外,HTTP API可以用来动态添加,移除和修改检查,以及进行TTL检查.TTL可以让应用程序更紧密的与Consul集成.将检查的状态加入到业务逻辑的计算.

键值数据存储

除了提供服务发现和健康检查的集成.Consul提供了一个易用的键/值存储.这可以用来保持动态配置,协助服务协调,领袖选举,做开发者可以想到的任何事情.

这一章假设你已经有至少一个Consul的agent在运行.

简单使用

为了演示如果简单的使用键值存储.我们将操作一些键.查询本地agent我们首先确认现在还没有存储任何key.

```
[root@hdp3 consul.d]# curl -v http://localhost:8500/v1/kv/?recur
se
* About to connect() to localhost port 8500 (#0)
*   Trying ::1... 拒绝连接
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8500 (#0)
> GET /v1/kv/?recurse HTTP/1.1
> User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu) libcurl/7.19
.7 NSS/3.21 Basic ECC zlib/1.2.3 libidn/1.18 libssh2/1.4.2
> Host: localhost:8500
> Accept: */*
>
< HTTP/1.1 404 Not Found
< X-Consul-Index: 1
< X-Consul-Knownleader: true
< X-Consul-Lastcontact: 0
< Date: Thu, 18 Aug 2016 08:21:39 GMT
< Content-Length: 0
< Content-Type: text/plain; charset=utf-8
<
* Connection #0 to host localhost left intact
* Closing connection #0
```

因为没有key所以我们得到了一个404响应.现在我们 ``PUT` 一些示例的Key:

```
[root@hdp3 consul.d]# curl -X PUT -d 'test' http://localhost:850
0/v1/kv/web/key1
[root@hdp3 consul.d]# curl -X PUT -d 'test' http://localhost:850
0/v1/kv/web/key2?flags=42
[root@hdp3 consul.d]# curl -X PUT -d 'test'  http://localhost:85
00/v1/kv/web/sub/key3
[root@hdp3 consul.d]# curl http://localhost:8500/v1/kv/?recurse
[{"LockIndex":0,"Key":"web/key1","Flags":0,"Value":"dGVzdA==","C
reateIndex":1201,"ModifyIndex":1201},{"LockIndex":0,"Key":"web/k
ey2","Flags":42,"Value":"dGVzdA==","CreateIndex":1205,"ModifyInd
ex":1206},{"LockIndex":0,"Key":"web/sub/key3","Flags":0,"Value":
"dGVzdA==","CreateIndex":1217,"ModifyIndex":1217}]
```

我们创建了值为"test"的3个Key,注意返回的值是经过了base64编码的.用来支持非UTF8编码字符.对Key `web/key2` 我们设置了一个标志值为 `42` .所有的key支持设置一个64位的整形数字标志.Consul内部不适用这个值.但是他可以被客户端适用来做一些元数据.

完成设置后,我们发起了一个 `GET` 请求来接收多个key的值,使用 `?recurse` 参数.

你可以获取单个的key

```
[root@hdp3 consul.d]# curl http://localhost:8500/v1/kv/web/key1
[{"LockIndex":0,"Key":"web/key1","Flags":0,"Value":"dGVzdA==","C
reateIndex":1201,"ModifyIndex":1201}]
```

删除key也很简单.通过 `DELETE` 动作来完成.我们可以通过指定完整路径来删除一个单独的key.或者我们可以使用 `?recurse` 递归的删除主路径下所有key.

```
[root@hdp3 consul.d]# curl -X DELETE http://localhost:8500/v1/kv
/web/sub?recurse
true
[root@hdp3 consul.d]# curl http://localhost:8500/v1/kv/web?recur
se
[{"LockIndex":0,"Key":"web/key1","Flags":0,"Value":"dGVzdA==","C
reateIndex":1201,"ModifyIndex":1201},{"LockIndex":0,"Key":"web/k
ey2","Flags":42,"Value":"dGVzdA==","CreateIndex":1205,"ModifyInd
ex":1206}]
```

可以通过发送相同的URL并提供不同的消息体的 `PUT` 请求去修改一个Key.另外,Consul提供一个检查并设置的操作,实现原子的Key修改.通过 `?cas=` 参数加上 `GET` 中最近的 `ModifyIndex` 来达到. 例如我们想修改 "web/key1":

```
[root@hdp3 consul.d]# curl -X PUT -d 'newval' http://localhost:8
500/v1/kv/web/key1?cas=1201
true
[root@hdp3 consul.d]# curl -X PUT -d 'newval' http://localhost:8
500/v1/kv/web/key1?cas=1201
false
```

在这种情况下,第一次 `CAS` 更新成功因为 `ModifyIndex` 是 `1201` .而第二次失败是因为 `ModifyIndex` 在第一次更新后已经不是 `1201` 了 .

我们也可以使用 `ModifyIndex` 来等待key值的改变.例如我们想等待 `key2` 被修改:

```
[root@hdp3 consul.d]# curl http://localhost:8500/v1/kv/web/key2
[{"LockIndex":0,"Key":"web/key2","Flags":42,"Value":"dGVzdA==","
CreateIndex":1205,"ModifyIndex":1206}]

[root@hdp3 consul.d]# curl "http://localhost:8500/v1/kv/web/key2
?index=1206&wait=5s"
[{"LockIndex":0,"Key":"web/key2","Flags":42,"Value":"dGVzdA==","
CreateIndex":1205,"ModifyIndex":1206}]
```

通过提供 `?index=` ,我们请求等待key值有一个比 `1206` 更大的 `ModifyIndex` .虽然 `?wait=5s` 参数限制了这个请求最多5秒,否则返回当前的未改变的值. 这样可以有效的等待key的改变.另外,这个功能可以用于等待一组key.直到其中的某个key有修改.

下一步

这里有一些说API可以支持的操作的例子,要查看完整文档.请查看这里.

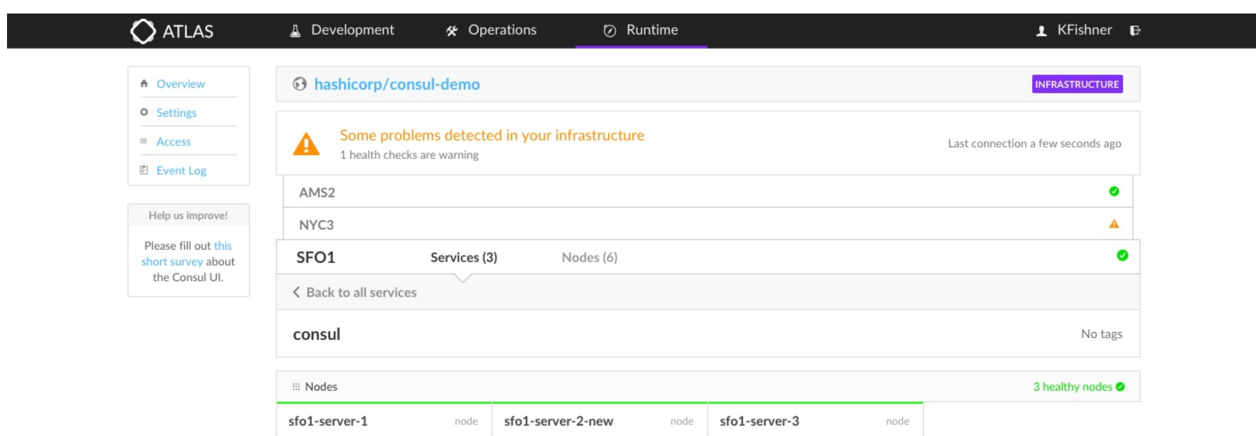
下面我们将看一看Consul支持的WebUI选项.

WEB界面

Consul同时提供了一个漂亮的功能齐全的WEB界面,开箱即用.界面可以用来查看所有的节点,可以查看健康检查和他们的当前状态.可以读取和设置K/V 存储的数据.UI自动支持多数据中心.

运行WebUI有两个选项.使用HashiCorp提供的Atlas来托管你的仪表盘或者使用Consul自己托管的开源UI

Atlas托管的仪表盘

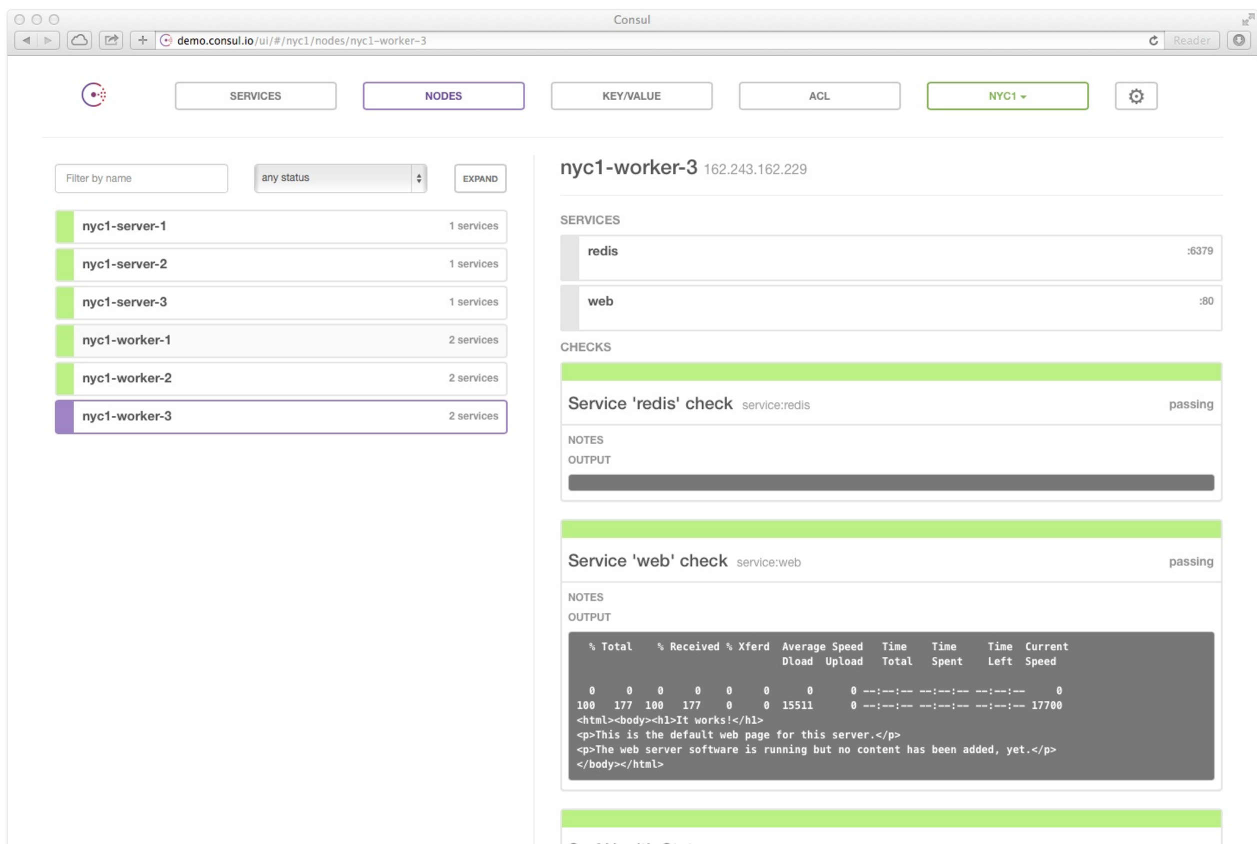


为了设置Consul使用Atlas界面.你必须添加两个字段到你的配置文件:你的Atlas名称和TOKEN.下面是一个命令行示例用来配置agent的这些设置:

```
$ consul agent -atlas=ATLAS_USERNAME/demo -atlas-token="ATLAS_TO
KEN"
```

获取Atlas用户名和token,创建一个账号并替换成你自己的配置.你可以查看线上Demo.

Consul托管的仪表盘



设置自托管的UI服务,启动Consul时使用 `-ui` 参数:

```
[root@hdp4 ~]# consul agent -ui
```

UI的路径在 `ui` ,使用HTTP API 相同的端口.默认为 `http://localhost:8500/ui` .

>

> 译者注: `-bind` 指定你要将HTTP绑定到的IP,绑定到一个公网IP一边可以从外部访问,否则只能在本机进行访问.所以我的启动命令是 `[root@hdp4 ~]# consul agent -ui -data-dir /tmp/consul -bind 10.0.`

你可以查看线上Demo.

线上Demo可以访问所有的数据中心.我们也设置了Demo的端点: AMS2 (阿姆斯特丹) , SF01(旧金山) 和 NY3(纽约).

下一步

我们的入门指南完成了.查看下一页来了解如何继续你与Consul的旅程!

接下来的

入门指南结束了,希望你能看到Consul是简单易用的.他拥有一些强大的功能.我们在这个指南里覆盖了所有这些功能的基础.

Consul采用对运维和开发者友好的方式进行设计,让它完美的适用于现代的弹性基础设施.

接下来可以使用这些资源做更深入的了解.

- 文档 - 文档是更深入的Consul功能参考.包括Consul内部操作的技术细节
- 指南 - 这部分提供很多Consul的入门指南,包括如何启动一个数据中心.
- 示例 - 这个Consul的Github库中还在进行中的示例目录包含很多使用示例,帮助你更好的根据你的需要使用Consul的功能.

Consul Template 简介

Consul Template 提供一个方便的方式从Consul获取数据通过consul-template的后台程序保存到文件系统.

这个后台进程监控Consul示例的变化并更新任意数量的模板到文件系统.作为一个附件功能,模板更新完成后consul-template可以运行任何命令.可以查看示例部分看这个功能将会对哪些应用场景产生帮助.

安装

你可以在发布页下载发布包.如果你希望自己编译请查看说明文档.

使用

选项

查看全部选项,使用以下命令

```
consul-template -h
```

命令行

```
查询 ```demo.consul.io``` 这个 ```Consul```实例(agent).渲染模板文件 ```/tmp/template.ctmpl``` 保存到 ``` /tmp/result```, 运行```Consul-template``` 服务直到直到手动结束:
```

consul-template \ -consul demo.consul.io \ -template "/tmp/template.ctmpl:/tmp/result"

```
查询本地的```Consul```实例(agent),一旦模板发生变化渲染模板并重启```Nginx```,如果```Consul```不可用30秒重试一次:
```

consul-template \ -consul 127.0.0.1:8500 \ -template "/tmp/template.ctmpl:/var/www/nginx.conf:service nginx restart" \ -retry 30s \ -once

```
查询一个```Consul```实例,渲染多个模板并作为服务直到停止:
```

consul-template \ -consul my.consul.internal:6124 \ -template "/tmp/nginx.ctmpl:/var/nginx/nginx.conf:service nginx restart" \ -template "/tmp/redis.ctmpl:/var/redis/redis.conf:service redis restart" \ -template "/tmp/haproxy.ctmpl:/var/haproxy/haproxy.conf"

```
查询一个需要权限验证的```Consul```实例,将渲染后的模板输出到控制台而不写入
磁盘.在这个例子中```-template```的第二个和第三个参数是必须的但是被忽略了.
这个文件将不会被写入磁盘,命令也不会被执行.
```

$ consul-template \ -consul my.consul.internal:6124 \ -template "/tmp/template.ctmpl:/tmp/result:service nginx restart" -dry

```
使用SSL证书进行```Consul```的查询:
```

$ consul-template \ -consul 127.0.0.1:8543 \ -ssl \ -ssl-cert /path/to/client/cert.pem \ -ssl-ca-cert /path/to/ca/cert.pem \ -template "/tmp/template.ctmpl:/tmp/result" \ -dry \ -once

```
查询```Consul```并启动一个子进程.模板的变化会发送指令给子进程.详细的说明请
查看[这里](https://github.com/hashicorp/consul-template#exec-mode
).
```

$ consul-template \ -template "/tmp/in.ctmpl:/tmp/result" \ -exec "/sbin/my-server"

```


#### 配置文件

```Consul-Template```配置文件是使用[HashiCorp Configuration Langua
ge (HCL)](https://github.com/hashicorp/hcl)编写的.这意味着```Consu
l Template```是和JSON兼容的,查看更多信息请查看 [HCL 规范](https://git
hub.com/hashicorp/hcl)

配置文件语法支持上面的所有的选项,除非在表格中进行标明.

```json

// 这是要连接的Consul Agent的地址.默认为127.0.0.1:8500.这是Consul的默
认绑定地址和端口.
```

```
// 不建议你直接与 Consul的 Server直接进行交互,请与本地的Consul Agent进行交互.这样做是有一些原因
// 最重要的是本地agent可以复用与server的连接.减少HTTP的连接数.另外这个地址更好记.
consul = "127.0.0.1:8500"

// 这是用于连接Consul的ACL token. 如果你的集群未启用就不需要设置.
//
// 这个选项也可以通过环境变量 CONSUL_TOKEN 来进行设置
token = "abcd1234"

// 这是监听出发reload事件的信号,默认值如下所示.将这个值设置为空将引起 CT ,从而不监听reload事件
reload_signal = "SIGHUP"

// 这是监听出发core dump事件的信号,默认值如下所示.将这个值设置为空将引起 CT ,从而不监听core dump信号
dump_signal = "SIGQUIT"

// 这是监听出发graceful stop事件的信号,默认值如下所示.将这个值设置为空将引起 CT ,从而不监听graceful stop信号
kill_signal = "SIGINT"

// 这是连接Consul的重试时间.Consul Template是高容错的设计.这意味着,出现失败他不会退出.而按照
// 分布式系统的惯例进行指数补偿和重试来等待集群恢复.
retry = "10s"

// This is the maximum interval to allow "stale" data. By default, only the
// Consul leader will respond to queries; any requests to a follower will
// forward to the leader. In large clusters with many requests, this is not as
// scalable, so this option allows any follower to respond to a query, so long
// as the last-replicated data is within these bounds. Higher values result in
// less cluster load, but are more likely to have outdated data.
// 这是允许陈旧数据的最大时间.Consul默认只有领袖对请求进行相应.所有对追随者的请求将被转发给领袖.
// 在有大量请求的大型集群中,这显得不够有扩展性.所以这个选项允许任何追随者响应查询,只要最后复制的数据
// 在这个范围内.数值越高,越减少集群负载,但是更容易接受到过期数据.
max_stale = "10m"

// 这是log的等级,如果你找到了bug,请打开debug 日志,这样我们可以更好的定位问题.这个选项也可用在命令行.
log_level = "warn"

// 这是存放Consul Template 进程的PID文件的路径,如果你计划发送定制的信号到这个进程这会比较有用.
pid_file = "/path/to/pid"
```

```
// 这是一个静止定时器,他定义了在模板渲染之前等待集群达到一致状态的最小和最大时间.
// 这对于一些变化较大的系统中比较有用,可以减少模板渲染的次数
wait = "5s:10s"


// 这是 Vault配置的开始
// Vault是HashiCorp的另外一个产品
vault {
  // This is the address of the Vault leader. The protocol (http(s)) portion
  // of the address is required.
  address = "https://vault.service.consul:8200"

  // This is the token to use when communicating with the Vault server.
  // Like other tools that integrate with Vault, Consul Template makes the
  // assumption that you provide it with a Vault token; it does not have the
  // incorporated logic to generate tokens via Vault's auth methods.
  //
  // This value can also be specified via the environment variable VAULT_TOKEN.
  token = "abcd1234"

  // This option tells Consul Template to automatically renew the Vault token
  // given. If you are unfamiliar with Vault's architecture, Vault requires
  // tokens be renewed at some regular interval or they will be revoked. Consul
  // Template will automatically renew the token at half the lease duration of
  // the token. The default value is true, but this option can be disabled if
  // you want to renew the Vault token using an out-of-band process.
  //
  // Note that secrets specified in a template (using {{secret}} for example)
  // are always renewed, even if this option is set to false. This option only
  // applies to the top-level Vault token itself.
  renew = true

  // This section details the SSL options for connecting to the Vault server.
  // Please see the SSL options below for more information (they are the same).
  ssl {
```

```
    // ...
  }
}

// 这部分配置请求的基本的权限验证信息
auth {
  enabled  = true
  username = "test"
  password = "test"
}

// 这部分配置连接到Consul服务器的SSL信息.
ssl {
  // 使用SSL需要先打开这个开关
  enabled = true

  // This enables SSL peer verification. The default value is "t
rue", which
  // will check the global CA chain to make sure the given certi
ficates are
  // valid. If you are using a self-signed certificate that you
have not added
  // to the CA chain, you may want to disable SSL verification.
However, please
  // understand this is a potential security vulnerability.
  verify = false

  // This is the path to the certificate to use to authenticate.
 If just a
  // certificate is provided, it is assumed to contain both the
certificate and
  // the key to convert to an X509 certificate. If both the cert
ificate and
  // key are specified, Consul Template will automatically combi
ne them into an
  // X509 certificate for you.
  cert = "/path/to/client/cert"
  key = "/path/to/client/key"

  // This is the path to the certificate authority to use as a C
A. This is
  // useful for self-signed certificates or for organizations us
ing their own
  // internal certificate authority.
  ca_cert = "/path/to/ca"
}

// 设置连接到syslog服务器的配置
// 用于进行日志记录syslog {
  // 打开开关
  enabled = true

  // 设备名称
```

```
  facility = "LOCAL5"
}

// This block defines the configuration for de-duplication mode.
 Please see the
// de-duplication mode documentation later in the README for mor
e information
// on how de-duplication mode operates.
deduplicate {
  // This enables de-duplication mode. Specifying any other opti
ons also enables
  // de-duplication mode.
  enabled = true

  // This is the prefix to the path in Consul's KV store where d
e-duplication
  // templates will be pre-rendered and stored.
  prefix = "consul-template/dedup/"
}

// This block defines the configuration for exec mode. Please se
e the exec mode
// documentation at the bottom of this README for more informati
on on how exec
// mode operates and the caveats of this mode.
exec {
  // This is the command to exec as a child process. There can b
e only one
  // command per Consul Template process.
  command = "/usr/bin/app"

  // This is a random splay to wait before killing the command.
The default
  // value is 0 (no wait), but large clusters should consider se
tting a splay
  // value to prevent all child processes from reloading at the
same time when
  // data changes occur. When this value is set to non-zero, Con
sul Template
  // will wait a random period of time up to the splay value bef
ore reloading
  // or killing the child process. This can be used to prevent t
he thundering
  // herd problem on applications that do not gracefully reload.
  splay = "5s"

  // This defines the signal that will be sent to the child proc
ess when a
  // change occurs in a watched template. The signal will only b
e sent after
  // the process is started, and the process will only be starte
d after all
  // dependent templates have been rendered at least once. The d
```

```
efault value
  // is "" (empty or nil), which tells Consul Template to restar
t the child
  // process instead of sending it a signal. This is useful for
legacy
  // applications or applications that cannot properly reload th
eir
  // configuration without a full reload.
  reload_signal = "SIGUSR1"

  // This defines the signal sent to the child process when Cons
ul Template is
  // gracefully shutting down. The application should begin a gr
aceful cleanup.
  // If the application does not terminate before the `kill_time
out`, it will
  // be terminated (effectively "kill -9"). The default value is
 "SIGTERM".
  kill_signal = "SIGINT"

  // This defines the amount of time to wait for the child proce
ss to gracefully
  // terminate when Consul Template exits. After this specified
time, the child
  // process will be force-killed (effectively "kill -9"). The d
efault value is
  // "30s".
  kill_timeout = "2s"
}

// 这部分定义了对模板的配置,和其他配置块不同.这部分可以针对不同模板配置多次.
也可以在CLI命令
// 直接进行配置
template {
  // 这是输入模板的配置文件路径,必选项
  source = "/path/on/disk/to/template.ctmpl"

  // 这是源模板渲染之后存放的路径,如果父目录不存在Consul Template会尝试进
行创建
  destination = "/path/on/disk/where/template/will/render.txt"

  // This is the optional command to run when the template is re
ndered. The
  // command will only run if the resulting template changes. Th
e command must
  // return within 30s (configurable), and it must have a succes
sful exit code.
  // Consul Template is not a replacement for a process monitor
or init system.
  // 这是当模板渲染完成后可选的要执行的命令.这个命令只会在模板发生改变后才会
运行.这个命令必须要在30秒
  // 内进行返回(可配置),必须返回一个成功的退出码.Consul Template不能替代
进程监视或者init 系统
```

```
  // 的功能
  command = "restart service foo"

  // 这是最大的等待命令返回的时间,默认是30秒
  command_timeout = "60s"

  // 这是渲染后的文件的权限,如果不设置,Consul Template将去匹配之前已经存在的文件的权限.
  // 如果文件不存在,权限会被设置为 0644
  perms = 0600

  // 这个选项对渲染之前的文件进行备份.他保持一个备份.
  // 这个选项在发生意外更高时,有一个回滚策略.
  backup = true

  // 模板的分隔符,默认是 "{{"和"}}".但是对于一些模板用其他的分隔符可能更好
  // 可以避免与本身的冲突
  left_delimiter  = "{{"
  right_delimiter = "}}"

  // 这是最小和最大等待渲染一个新模板和执行命令的时间.使用 分号 个号.如果忽略最大值,最大
  // 值会被设置为最小值的4倍.这个选项没有默认值.这个值相对全局所以的等待时间有最高优先级
  wait = "2s:6s"
}
```

> 注意: 不是所有的选项都是必选的.例如: 如果你没有使用Vault你不用设置这一块. 类似的你没有使用syslog系统你也不需要指定syslog配置.

为了更加安全, `token` 也可以从环境变量里读取,使用 `CONSUL_TOKEN` 和 `VAULT_TOKEN` .强烈建议你不要把token放到未加密的文本配置文件中.

查询 nyc3 demo 的Consul示例, 渲染模板 `/tmp/template.ctmpl` 到 `/tmp/result` .运行Consul Template直到服务停止:

```
consul = "nyc3.demo.consul.io"

template {
  source      = "/tmp/template.ctmpl"
  destination = "/tmp/result"
}
```

如果一个用一个目录替换文件,所以这个目录中的文件会递归的安装Go walk函数的顺序进行合并.所以如果多个文件定义了 `consul` key 则最后一个将会被使用,注意,符号链接不会被加入.

在命令行指定的选项,优先于配置文件

模板语法

Consul Template 使用了Go的模板语法.如果你对他的语法不熟悉建议你读下文档.他的语法看起来与 Mustache, Handlebars, 或者 Liquid 类似.

在Go 提供的模板函数之外,Consul Template暴露了以下的函数:

API 函数

datacenters

查询目录中的所有数据中心.使用以下语法:

```
{{datacenters}}
```

file

读取并输出磁盘上的本地文件,如果无法读取产生一个错误.使用如下语法

```
{{file "/path/to/local/file"}}
```

这个例子将输出 `/path/to/local/file` 文件内容到模板. 注意:这不会在嵌套模板中被处理

key

查询Consul指定key的值,如果key的值不能转换为字符串,将产生错误.使用如下语法:

```
{{key "service/redis/maxconns@east-aws"}}
```

上面的例子查询了在`east-aws`数据中心的 `service/redis/maxconns` 的值.如果忽略数据中心参数,将会查询本地数据中心的值:

```
{{key "service/redis/maxconns"}}
```

Consul键值结构的美妙在于,这完全取决于你!

key_or_default

查询Consul中指定的key的值,如果key不存在,则返回默认值.使用方式如下

```
{{key_or_default "service/redis/maxconns@east-aws" "5"}}
```

注意Consul Template使用了多个阶段的运算.在第一阶段的运算如果Consul没有返回值,则会一直使用默认值.后续模板解析中如果值存在了则会读取真实的值.这很重要,运维Consul Templae不会因为 `key_or_default` 没找到key而阻塞模板的的渲染.即使key存在如果Consul没有按时返回这个数据,也会使用默认值来进行替代.

ls

查看Consul的所有以指定前缀开头的key-value对.如果有值无法转换成字符串则会产生一个错误:

```
{{range ls "service/redis@east-aws"}}
{{.Key}} {{.Value}}{{end}}
```

如果Consul实例在 `east-aws` 数据中心存在这个结构 `service/redis` ,渲染后的模板应该类似这样:

```
minconns 2
maxconns 12
```

如果你忽略数据中心属性,则会返回本地数据中心的查询结果.

node

查询目录中的一个节点信息

```
{{node "node1"}}
```

如果未指定任何参数,则当前agent所在节点将会被返回:

```
{{node}}
```

你可以指定一个可选的参数来指定数据中心:

```
{{node "node1" "@east-aws"}}
```

如果指定的节点没有找到则会返回 `nil` .如果节点存在就会列出节点的信息和节点提供的服务.

```
{{with node}}{{.Node.Node}} ({{.Node.Address}}){{range .Services
}}
  {{.Service}} {{.Port}} ({{.Tags | join ","}}){{end}}
{{end}}
```

nodes

查询目录中的全部节点,使用如下语法

```
{{nodes}}
```

这个例子会查询Consul的默认数据中心.你可以使用可选参数指定一个可选参数来指定数据中心:

```
{{nodes "@east-aws"}}
```

这个例子会查询 `east-aws` 数据中心的所有几点.

secret

查询 `Vault` 中指定路径的密匙.如果指定的路径不存在或者 `Vault` 的Token没有足够权限去读取指定的路径,将会产生一个错误.如果路径存在但是key不存在则返回 `<no value>` .

```
{{with secret "secret/passwords"}}{{.Data.password}}{{end}}
```

可以使用如下字段:

```
LeaseID - the unique lease identifier
LeaseDuration - the number of seconds the lease is valid
Renewable - if the secret is renewable
Data - the raw data - this is a map[string]interface{}, so it ca
n be queried using Go's templating "dot notation"
If the map key has dots "." in it, you need to access the value
using the index function:

{{index .Data "my.key.with.dots"}}
If additional arguments are passed to the function, then the ope
ration is assumed to be a write operation instead of a read oper
ation. The write operation must return data in order to be valid
. This is especially useful for the PKI secret backend, for exam
ple.

{{ with secret "pki/issue/my-domain-dot-com" "common_name=foo.ex
ample.com" }}
{{ .Data.certificate }}
{{ end }}
The parameters must be key=value pairs, and each pair must be it
s own argument to the function:

{{ secret "path/" "a=b" "c=d" "e=f" }}
Please always consider the security implications of having the c
ontents of a secret in plain-text on disk. If an attacker is abl
e to get access to the file, they will have access to plain-text
 secrets.
```

Please note that Vault does not support blocking queries. As a result, Consul Template will not immediately reload in the event a secret is changed as it does with Consul's key-value store. Consul Template will fetch a new secret at half the lease duration of the original secret. For example, most items in Vault's generic secret backend have a default 30 day lease. This means Consul Template will renew the secret every 15 days. As such, it is recommended that a smaller lease duration be used when generating the initial secret to force Consul Template to renew more often.

secrets

Query Vault to list the secrets at the given path. Please note this requires Vault 0.5+ and the endpoint you want to list secrets must support listing. Not all endpoints support listing. The result is the list of secret names as strings.

```
{{range secrets "secret/"}}{{.}}{{end}}
```

The trailing slash is optional in the template, but the generated secret dependency will always have a trailing slash in log output.

To iterate and list over every secret in the generic secret backend in Vault, for example, you would need to do something like this:

```
{{range secrets "secret/"}}
{{with secret (printf "secret/%s" .)}}
{{range $k, $v := .Data}}
{{$k}}: {{$v}}
{{end}}
{{end}}
{{end}}
```

You should probably never do this. Please also note that Vault does not support blocking queries. To understand the implications, please read the note at the end of the secret function.

service

查询Consul中匹配表达式的服务.语法如下:

```
{{service "release.web@east-aws"}}
```

上面的例子查询Consul中,在 `east-aws` 数据中心存在的健康的 `web` 服务.tag和数据中心参数是可选的.从当前数据中心查询所有节点的 `web` 服务而不管tag,查询语法如下:

```
{{service "web"}}
```

这个函数返回 `[]*HealthService` 结构.可按照如下方式应用到模板:

```
{{range service "web@data center"}}
server {{.Name}} {{.Address}}:{{.Port}}{{end}}
```

产生如下输出:

```
server nyc_web_01 123.456.789.10:8080
server nyc_web_02 456.789.101.213:8080
```

默认值会返回健康的服务,如果你想获取所有服务,可以增加 `any` 选项,如下:

```
{{service "web" "any"}}
```

这样就会返回注册过的所有服务,而不论他的状态如何.

如果你想过滤指定的一个或者多个健康状态,你可以通过逗号隔开多个健康状态:

```
{{service "web" "passing, warning"}}
```

这样将会返回被他们的节点和服务级别的检查定义标记为 "passing" 或者 "warning"的服务. 请注意逗号是 `OR` 而不是 `AND` 的意思.

指定了超过一个状态过滤,并包含 `any` 将会返回一个错误.因为 `any` 是比所有状态更高级的过滤.

后面这2种方式有些架构上的不同:

```
{{service "web"}}
{{service "web" "passing"}}
```

前者会返回Consul认为 `healthy` 和 `passing` 的所有服务.后者将返回所有已经在Consul注册的服务.然后会执行一个客户端的过滤.通常如果你想获取健康的服务,你应该不要使用 `passing` 参数,直接忽略第三个参数即可.然而第三个参数在你想查询 `passing` 或者 `warning` 的服务会比较有用,如下:

```
{{service "web" "passing, warning"}}
```

服务的状态也是可见的,如果你想自己做一些额外的过滤,语法如下:

```
{{range service "web" "any"}}
{{if eq .Status "critical"}}
// Critical state!{{end}}
{{if eq .Status "passing"}}
// Ok{{end}}
```

执行命令时,在Consul将服务设置为维护模式,只需要在你的命令上包上Consul的 `maint` 调用:

```
#!/bin/sh
set -e
consul maint -enable -service web -reason "Consul Template updat
ed"
service nginx reload
consul maint -disable -service web
```

另外如果你没有安装Consul agent,你可以直接调用API请求:

```
#!/bin/sh
set -e
curl -X PUT "http://$CONSUL_HTTP_ADDR/v1/agent/service/maintenan
ce/web?enable=true&reason=Consul+Template+Updated"
service nginx reload
curl -X PUT "http://$CONSUL_HTTP_ADDR/v1/agent/service/maintenan
ce/web?enable=false"
```

services

查询Consul目录中的所有服务,使用如下语法:

```
{{services}}
```

这个例子将查询Consul的默认数据中心,你可以指定一个可选参数来指定数据中心:

```
{{services "@east-aws"}}
```

请注意: `services` 函数与 `service` 是不同的, `service` 接受更多参数并且查询监控的服务列表.这个查询Consul目录并返回一个服务的tag的Map,如下:

```
{{range services}}
{{.Name}}
{{range .Tags}}
  {{.}}{{end}}
{{end}}
```

tree

查询所有指定前缀的key-value值对,如果其中的值有无法转换为字符串的则引发错误:

```
{{range tree "service/redis@east-aws"}}
{{.Key}} {{.Value}}{{end}}
```

如果Consul实例在 `east-aws` 数据中心有一个 `service/redis` 结构,模板的渲染结果类似下面:

```
minconns 2
maxconns 12
nested/config/value "value"
```

和 `ls` 不同, `tree` 返回前缀下的所有key.和Unix的tree命令比较像.如果忽略数据中心参数,则会使用本地数据中心

帮助函数

byKey

将 `tree` 返回的key-value值对结果创建一个map,这个map根据他们的顶级目录进行组合.例如如果Consul的kv存储如下结构:

```
groups/elasticsearch/es1
groups/elasticsearch/es2
groups/elasticsearch/es3
services/elasticsearch/check_elasticsearch
services/elasticsearch/check_indexes
```

使用下面的模板:

```
{{range $key, $pairs := tree "groups" | byKey}}{{$key}}:
{{range $pair := $pairs}}  {{.Key}}={{.Value}}
{{end}}{{end}}
```

结果如下:

```
elasticsearch:
  es1=1
  es2=1
  es3=1
```

注意顶部的key会被从key的值中剥离出来.如果在剥离前缀后没有前缀,值会被从列表移除.

结果的对会被映射为map,使用可以使用key来访问一个单独的值:

```
{{$weights := tree "weights"}}
{{range service "release.web"}}
  {{$weight := or (index $weights .Node) 100}}
  server {{.Node}} {{.Address}}:{{.Port}} weight {{$weight}}{{en
d}}
```

byTag

将被 `service` 或者 `services` 函数返回的列表,按照tag对服务创建Map.

```
{{range $tag, $services := service "web" | byTag}}{{$tag}}
{{range $services}} server {{.Name}} {{.Address}}:{{.Port}}
{{end}}{{end}}
```

contains

检查目标是否包含在枚举的元素中

```
{{ if .Tags | contains "production" }}
# ...
{{ end }}
env
```

读取当前进程可以访问的环境变量.

```
{{env "CLUSTER_ID"}}
```

这个函数可以加入管道:

```
{{env "CLUSTER_ID" | toLower}}
```

explode

将 `tree` 或者 `ls` 的结果转化为深度嵌套的map,用来进行解析和递归.

```
{{ tree "config" | explode }}
```

注意: 解开后,你将丢失所有的关于键值对的元数据.

你可以访问深度嵌套的值:

```
{{ with tree "config" | explode }}
{{.a.b.c}}{{ end }}
```

注意: 你需要在Consul中保存有一个合理的格式的数据.可以查看Go的 text/template包获取更多信息.

in

检查目标十分在一个可枚举的元素中.

```
{ if in .Tags "production" }}
# ...
{{ end }}
```

loop

接受多个参数,行为受这些参数的影响.

如果给 `loop` 一个数字,他讲返回一个 `goroutine` ,开始于0循环直到等于参数的值:

```
{{range loop 5}}
# Comment{{end}}
```

如果给2个数字,则这个函数返回一个 `goroutine` 从第一个数字开始循环直到等于第二个参数的值.

```
{{range $i := loop 5 8}}
stanza-{{$i}}{{end}}
```

渲染结果为:

```
stanza-5
stanza-6
stanza-7
```

Note: It is not possible to get the index and the element since the function returns a goroutine, not a slice. In other words, the following is not valid:

```
# Will NOT work!
{{range $i, $e := loop 5 8}}
# ...{{end}}
```

join

将提供的列表作为管道与提供的字符串连接:

```
{{$items | join ","}}
```

trimSpace

对输入的内容移除掉空白,tab和换行符

```
{ file "/etc/ec2_version"| trimSpace }}
```

parseBool

将给定的字符串解析为布尔值:

```
{{"true" | parseBool}}
```

这个可以与一个key检查和条件检查相结合.如下:

```
{{if key "feature/enabled" | parseBool}}{{end}}
```

parseFloat

将给定的字符串解析为 10进制 float64类型数字:

```
{{"1.2" | parseFloat}}
```

parseInt

将给定字符串解析为10禁止 int64类型数字:

```
{{"1" | parseInt}}
```

这个可以与其他的帮助函数结合使用,例如:

```
{{range $i := loop key "config/pool_size" | parseInt}}
# ...{{end}}
```

parseJSON

将输入,通常是通过key获取的值,解析成JSON

Takes the given input (usually the value from a key) and parses the result as JSON:

```
{{with $d := key "user/info" | parseJSON}}{{$d.name}}{{end}}
```

注意 : Consul Template计算模板很多次.第一次计算时会是空,因为数据还未载入,这意味着我们需要检查空的响应.例如:

```
{{with $d := key "user/info" | parseJSON}}
{{if $d}}
...
{{end}}
{{end}}
```

它只适用简单的key,但是如果你想遍历key或者使用index函数会失败.将要访问的代码包含在 `{{ if $d }}...{{end}}` 之中就够了.

Alternatively you can read data from a local JSON file:

```
{{with $d := file "/path/to/local/data.json" | parseJSON}}{{$d.some_key}}{{end}}
```

parseUint

Takes the given string and parses it as a base-10 int64:

```
{{"1" | parseUint}}
```

See parseInt for examples.

plugin

Takes the name of a plugin and optional payload and executes a Consul Template plugin.

```
{{plugin "my-plugin"}}
```

This is most commonly combined with a JSON filter for customization:

```
{{tree "foo" | explode | toJSON | plugin "my-plugin"}}
```

Please see the plugins section for more information about plugins.

regexMatch

Takes the argument as a regular expression and will return true if it matches on the given string, or false otherwise.

```
{{"foo.bar" | regexMatch "foo([.a-z]+)"}}
```

regexReplaceAll

Takes the argument as a regular expression and replaces all occurrences of the regex with the given string. As in go, you can use variables like $1 to refer to subexpressions in the replacement string.

```
{{"foo.bar" | regexReplaceAll "foo([.a-z]+)" "$1"}}
```

replaceAll

Takes the argument as a string and replaces all occurrences of the given string with the given string.

```
{{"foo.bar" | replaceAll "." "_"}}
```

This function can be chained with other functions as well:

```
{{service "web"}}{{.Name | replaceAll ":" "_"}}{{end}}
```

split

Splits the given string on the provided separator:

```
{{"foo\nbar\n" | split "\n"}}
```

This can be combined with chained and piped with other functions:

```
{{key "foo" | toUpper | split "\n" | join ","}}
```

timestamp

Returns the current timestamp as a string (UTC). If no arguments are given, the result is the current RFC3339 timestamp:

```
{{timestamp}} // e.g. 1970-01-01T00:00:00Z
```

If the optional parameter is given, it is used to format the timestamp. The magic reference date Mon Jan 2 15:04:05 -0700 MST 2006 can be used to format the date as required:

```
{{timestamp "2006-01-02"}} // e.g. 1970-01-01
```

See Go's time.Format() for more information.

As a special case, if the optional parameter is "unix", the unix timestamp in seconds is returned as a string.

```
{{timestamp "unix"}} // e.g. 0
```

toJSON

Takes the result from a tree or ls call and converts it into a JSON object.

```
{{ tree "config" | explode | toJSON }} // e.g. {"admin":{"port":
1234},"maxconns":5,"minconns":2}
```

Note: This functionality should be considered final. If you need to manipulate keys, combine values, or perform mutations, that should be done outside of Consul. In order to keep the API scope limited, we likely will not accept Pull Requests that focus on customizing the toJSON functionality.

toJSONPretty

Takes the result from a tree or ls call and converts it into a pretty-printed JSON object, indented by two spaces.

```
{{ tree "config" | explode | toJSONPretty }}
```

```
/*
{
  "admin": {
    "port": 1234
  },
  "maxconns": 5,
  "minconns": 2,
}
*/
```

Note: This functionality should be considered final. If you need to manipulate keys, combine values, or perform mutations, that should be done outside of Consul. In order to keep the API scope limited, we likely will not accept Pull Requests that focus on customizing the toJSONPretty functionality.

toLower

Takes the argument as a string and converts it to lowercase.

```
{{key "user/name" | toLower}}
```

See Go's strings.ToLower() for more information.

toTitle

Takes the argument as a string and converts it to titlecase.

```
{{key "user/name" | toTitle}}
```

See Go's strings.Title() for more information.

toUpper

Takes the argument as a string and converts it to uppercase.

```
{{key "user/name" | toUpper}}
```

See Go's strings.ToUpper() for more information.

toYAML

Takes the result from a tree or ls call and converts it into a pretty-printed YAML object, indented by two spaces.

```
{{ tree "config" | explode | toYAML }}
/*
admin:
  port: 1234
maxconns: 5
minconns: 2
*/
```

Note: This functionality should be considered final. If you need to manipulate keys, combine values, or perform mutations, that should be done outside of Consul. In order to keep the API scope limited, we likely will not accept Pull Requests that focus on customizing the toYAML functionality.

Math Functions

The following functions are available on floats and integer values.

add

Returns the sum of the two values.

```
{{ add 1 2 }} // 3
```

This can also be used with a pipe function.

```
{{ 1 | add 2 }} // 3
```

subtract

Returns the difference of the second value from the first.

```
{{ subtract 2 5 }} // 3
```

This can also be used with a pipe function.

```
{{ 5 | subtract 2 }}
```

Please take careful note of the order of arguments.

multiply

Returns the product of the two values.

```
{{ multiply 2 2 }} // 4
```

This can also be used with a pipe function.

```
{{ 2 | multiply 2 }} // 4
```

divide

Returns the division of the second value from the first.

```
{{ divide 2 10 }} // 5
```

This can also be used with a pipe function.

```
{{ 10 | divide 2 }} // 5
```

Please take careful note of the order or arguments.

Plugins

Authoring Plugins

For some use cases, it may be necessary to write a plugin that offloads work to another system. This is especially useful for things that may not fit in the "standard library" of Consul Template, but still need to be shared across multiple instances.

Consul Template plugins must have the following API:

```
$ NAME [INPUT...]
```

NAME - the name of the plugin - this is also the name of the binary, either a full path or just the program name. It will be executed in a shell with the inherited PATH so e.g. the plugin cat will run the first executable cat that is found on the PATH. INPUT - input from the template - this will always be JSON if provided Important Notes

Plugins execute user-provided scripts and pass in potentially sensitive data from Consul or Vault. Nothing is validated or protected by Consul Template, so all necessary precautions and considerations should be made by template authors Plugin output must be returned as a string on stdout. Only stdout will be parsed for output. Be sure to log all errors, debugging messages onto stderr to avoid errors when Consul Template returns the value. Always exit 0 or Consul Template will assume the plugin failed to execute Ensure the empty input case is handled correctly (see Multi-phase execution) Data piped into the plugin is appended after any parameters given explicitly (eg `{{ "sample-data" | plugin "my-plugin" "some-parameter"}}` will call my-plugin some-parameter sample-data) Here is a sample plugin in a few different languages that removes any JSON keys that start with an underscore and returns the JSON string:

```
#! /usr/bin/env ruby
require "json"

if ARGV.empty?
  puts JSON.fast_generate({})
  Kernel.exit(0)
end

hash = JSON.parse(ARGV.first)
hash.reject! { |k, _| k.start_with?("_")  }
puts JSON.fast_generate(hash)
Kernel.exit(0)
func main() {
  arg := []byte(os.Args[1])

  var parsed map[string]interface{}
  if err := json.Unmarshal(arg, &parsed); err != nil {
    fmt.Fprintln(os.Stderr, fmt.Sprintf("err: %s", err))
    os.Exit(1)
  }

  for k, _ := range parsed {
    if string(k[0]) == "_" {
      delete(parsed, k)
    }
  }

  result, err := json.Marshal(parsed)
  if err != nil {
    fmt.Fprintln(os.Stderr, fmt.Sprintf("err: %s", err))
    os.Exit(1)
  }

  fmt.Fprintln(os.Stdout, fmt.Sprintf("%s", result))
  os.Exit(0)
}
Caveats
```

Exec Mode

As of version 0.16.0, Consul Template has the ability to maintain an arbitrary child process (similar to envconsul). This mode is most beneficial when running Consul Template in a container or on a scheduler like Nomad or Kubernetes. When activated, Consul Template will spawn and manage the lifecycle of the child process.

This mode is best-explained through example. Consider a simple application that reads a configuration file from disk and spawns a server from that configuration.

```
$ consul-template \
    -template="/tmp/config.ctmpl:/tmp/server.conf" \
    -exec="/bin/my-server -config /tmp/server.conf"
```

When Consul Template starts, it will pull the required dependencies and populate the /tmp/server.conf, which the my-server binary consumes. After that template is rendered completely the first time, Consul Template spawns and manages a child process. When any of the list templates change, Consul Template will send the configurable reload signal to that child process. If no reload signal is provided, Consul Template will kill and restart the process. Additionally, in this mode, Consul Template will proxy any signals it receives to the child process. This enables a scheduler to control the lifecycle of the process and also eases the friction of running inside a container.

A common point of confusion is that the command string behaves the same as the shell; it does not. In the shell, when you run foo | bar or foo > bar, that is actually running as a subprocess of your shell (bash, zsh, csh, etc.). When Consul Template spawns the exec process, it runs outside of your shell. This behavior is different from when Consul Template executes the template-specific reload command. If you want the ability to pipe or redirect in the exec command, you will need to spawn the process in subshell, for example:

exec { command = "$SHELL -c 'my-server > /var/log/my-server.log'" } Note that when spawning like this, most shells do not proxy signals to their child by default, so your child process will not receive the signals that Consul Template sends to the shell. You can avoid this by writing a tiny shell wrapper and executing that instead:

!/usr/bin/env bash

trap "kill -TERM $child" SIGTERM

/bin/my-server -config /tmp/server.conf child=$! wait "$child" Alternatively, you can use your shell's exec function directly, if it exists:

```
#!/usr/bin/env bash
exec /bin/my-server -config /tmp/server.conf > /var/log/my-serve
r.log
```

There are some additional caveats with Exec Mode, which should be considered carefully before use:

If the child process dies, the Consul Template process will also die. Consul Template does not supervise the process! This is generally the responsibility of the scheduler or init system. The child process must remain in the foreground. This is a requirement for Consul Template to manage the process and send

signals. The exec command will only start after all templates have been rendered at least once. One may have multiple templates for a single Consul Template process, all of which must be rendered before the process starts. Consider something like an nginx or apache configuration where both the process configuration file and individual site configuration must be written in order for the service to successfully start. After the child process is started, any change to any dependent template will cause the reload signal to be sent to the child process. This reload signal defaults to nil, in which Consul Template will not kill and respawn the child. The reload signal can be specified and customized via the CLI or configuration file. When Consul Template is stopped gracefully, it will send the configurable kill signal to the child process. The default value is SIGTERM, but it can be customized via the CLI or configuration file. Consul Template will forward all signals it receives to the child process except its defined reload_signal, dump_signal, and kill_signal. If you disable these signals, Consul Template will forward them to the child process. It is not possible to have more than one exec command (although each template can still have its own reload command). Individual template reload commands still fire independently of the exec command. De-Duplication Mode

Consul Template works by parsing templates to determine what data is needed and then watching Consul for any changes to that data. This allows Consul Template to efficiently re-render templates when a change occurs. However, if there are many instances of Consul Template rendering a common template there is a linear duplication of work as each instance is querying the same data.

To make this pattern more efficient Consul Template supports work de-duplication across instances. This can be enabled with the -dedup flag or via the deduplicate configuration block. Once enabled, Consul Template uses leader election on a per-template basis to have only a single node perform the queries. Results are shared among other instances rendering the same template by passing compressed data through the Consul K/V store.

Please note that no Vault data will be stored in the compressed template. Because ACLs around Vault are typically more closely controlled than those ACLs around Consul's KV, Consul Template will still request the secret from Vault on each iteration.

Termination on Error

By default Consul Template is highly fault-tolerant. If Consul is unreachable or a template changes, Consul Template will happily continue running. The only exception to this rule is if the optional command exits non-zero. In this case, Consul Template will also exit non-zero. The reason for this decision is so the user can easily configure something like Upstart or God to manage Consul Template as a service.

If you want Consul Template to continue watching for changes, even if the optional command argument fails, you can append || true to your command. For example:

```
$ consul-template \
  -template "in.ctmpl:out.file:service nginx restart || true"
```

In this example, even if the Nginx restart command returns non-zero, the overall function will still return an OK exit code; Consul Template will continue to run as a service. Additionally, if you have complex logic for restarting your service, you can intelligently choose when you want Consul Template to exit and when you want it to continue to watch for changes. For these types of complex scripts, we recommend using a custom sh or bash script instead of putting the logic directly in the consul-template command or configuration file.

Command Environment

The current processes environment is used when executing commands with the following additional environment variables:

```
CONSUL_HTTP_ADDR
CONSUL_HTTP_TOKEN
CONSUL_HTTP_AUTH
CONSUL_HTTP_SSL
CONSUL_HTTP_SSL_VERIFY
```

These environment variables are exported with their current values when the command executes. Other Consul tooling reads these environment variables, providing smooth integration with other Consul tools (like consul maint or consul lock). Additionally, exposing these environment variables gives power users the ability to further customize their command script.

Multi-phase Execution

Consul Template does an n-pass evaluation of templates, accumulating dependencies on each pass. This is required due to nested dependencies, such as:

```
{{range services}}
{{range service .Name}}
  {{.Address}}
{{end}}{{end}}
```

During the first pass, Consul Template does not know any of the services in Consul, so it has to perform a query. When those results are returned, the inner-loop is then evaluated with that result, potentially creating more queries and watches.

Because of this implementation, template functions need a default value that is an acceptable parameter to a range function (or similar), but does not actually execute the inner loop (which would cause a panic). This is important to mention because complex templates must account for the "empty" case. For example, the following will not work:

```
{{with index (service "foo") 0}}
# ...
{{end}}
```

This will raise an error like:

: error calling index: index out of range: 0 That is because, during the first evaluation of the template, the service key is returning an empty slice. You can account for this in your template like so:

```
{{if service "foo"}}
{{with index (service "foo") 0}}
{{.Node}}
{{ end }}
{{ end }}
```

This will still add the dependency to the list of watches, but Go will not evaluate the inner-if, avoiding the out-of-index error.

Examples

HAProxy

HAProxy is a very common load balancer. You can read more about the HAProxy configuration file syntax in the HAProxy documentation, but here is an example template for rendering an HAProxy configuration file with Consul Template:

```
global
    daemon
    maxconn {{key "service/haproxy/maxconn"}}

defaults
    mode {{key "service/haproxy/mode"}}{{range ls "service/hapro
xy/timeouts"}}
    timeout {{.Key}} {{.Value}}{{end}}

listen http-in
    bind *:8000{{range service "release.web"}}
    server {{.Node}} {{.Address}}:{{.Port}}{{end}}
```

Save this file to disk as haproxy.ctmpl and run the consul-template daemon:

```
$ consul-template \
  -consul demo.consul.io \
  -template haproxy.ctmpl:/etc/haproxy/haproxy.conf
  -dry
```

Depending on the state of the demo Consul instance, you could see the following output:

```
global
    daemon
    maxconn 4

defaults
    mode default
    timeout 5

listen http-in
    bind *:8000
    server nyc3-worker-2 104.131.109.224:80
    server nyc3-worker-3 104.131.59.59:80
    server nyc3-worker-1 104.131.86.92:80
```

For more information on how to save this result to disk or for the full list of functionality available inside a Consul template file, please consult the API documentation.

Varnish

Varnish is an common caching engine that can also act as a proxy. You can read more about the Varnish configuration file syntax in the Varnish documentation, but here is an example template for rendering a Varnish configuration file with Consul Template:

```
import directors;
{{range service "consul"}}
backend {{.Name}}_{{.ID}} {
    .host = "{{.Address}}";
    .port = "{{.Port}}";
}{{end}}

sub vcl_init {
  new bar = directors.round_robin();
{{range service "consul"}}
  bar.add_backend({{.Name}}_{{.ID}});{{end}}
}

sub vcl_recv {
  set req.backend_hint = bar.backend();
}
```

Save this file to disk as varnish.ctmpl and run the consul-template daemon:

```
$ consul-template \
  -consul demo.consul.io \
  -template varnish.ctmpl:/etc/varnish/varnish.conf \
  -dry
```

You should see the following output:

```
import directors;

backend consul_consul {
    .host = "104.131.109.106";
    .port = "8300";"
}

sub vcl_init {
  new bar = directors.round_robin();

  bar.add_backend(consul_consul);
}

sub vcl_recv {
  set req.backend_hint = bar.backend();
}
```

Apache httpd

Apache httpd is a popular web server. You can read more about the Apache httpd configuration file syntax in the Apache httpd documentation, but here is an example template for rendering part of an Apache httpd configuration file that is responsible for configuring a reverse proxy with dynamic end points based on service tags with Consul Template:

```
{{range $tag, $service := service "web" | byTag}}
# "{{$tag}}" api providers.
<Proxy balancer://{{$tag}}>
{{range $service}}  BalancerMember http://{{.Address}}:{{.Port}}
{{end}} ProxySet lbmethod=bybusyness
</Proxy>
Redirect permanent /api/{{$tag}} /api/{{$tag}}/
ProxyPass /api/{{$tag}}/ balancer://{{$tag}}/
ProxyPassReverse /api/{{$tag}}/ balancer://{{$tag}}/
{{end}}
```

Just like the previous examples, save this file to disk and run the consul-template daemon:

```
$ consul-template \
  -consul demo.consul.io \
  -template httpd.ctmpl:/etc/httpd/sites-available/balancer.conf
```

You should see output similar to the following:

```
# "frontend" api providers.
<Proxy balancer://frontend>
  BalancerMember http://104.131.109.106:8080
  BalancerMember http://104.131.109.113:8081
  ProxySet lbmethod=bybusyness
</Proxy>
Redirect permanent /api/frontend /api/frontend/
ProxyPass /api/frontend/ balancer://frontend/
ProxyPassReverse /api/frontend/ balancer://frontend/

# "api" api providers.
<Proxy balancer://api>
  BalancerMember http://104.131.108.11:8500
  ProxySet lbmethod=bybusyness
</Proxy>
Redirect permanent /api/api /api/api/
ProxyPass /api/api/ balancer://api/
ProxyPassReverse /api/api/ balancer://api/
```

Querying all services

As of Consul Template 0.6.0, it is possible to have a complex dependency graph with dependent services. As such, it is possible to query and watch all services in Consul:

```
{{range services}}# {{.Name}}{{range service .Name}}
{{.Address}}{{end}}

{{end}}
```

Just like the previous examples, save this file to disk and run the consul-template daemon:

```
$ consul-template \
  -consul demo.consul.io \
  -template everything.ctmpl:/tmp/inventory
```

You should see output similar to the following:

```
# consul
104.131.121.232

# redis
104.131.86.92
104.131.109.224
104.131.59.59

# web
104.131.86.92
104.131.109.224
104.131.59.59
```

Running and Process Lifecycle

While there are multiple ways to run Consul Template, the most common pattern is to run Consul Template as a system service. When Consul Template first starts, it reads any configuration files and templates from disk and loads them into memory. From that point forward, changes to the files on disk do not propagate to running process without a reload.

The reason for this behavior is simple and aligns with other tools like haproxy. A user may want to perform pre-flight validation checks on the configuration or templates before loading them into the process. Additionally, a user may want to update configuration and templates simultaneously. Having Consul Template automatically watch and reload those files on changes is both operationally dangerous and against some of the paradigms of modern infrastructure. Instead, Consul Template listens for the SIGHUP syscall to trigger a configuration reload. If

you update configuration or templates, simply send HUP to the running Consul Template process and Consul Template will reload all the configurations and templates from disk.

Debugging

Consul Template can print verbose debugging output. To set the log level for Consul Template, use the -log-level flag:

```
$ consul-template -log-level info ...
<timestamp> [INFO] (cli) received redis from Watcher
<timestamp> [INFO] (cli) invoking Runner
# ...
```

You can also specify the level as debug:

```
$ consul-template -log-level debug ...
<timestamp> [DEBUG] (cli) creating Runner
<timestamp> [DEBUG] (cli) creating Consul API client
<timestamp> [DEBUG] (cli) creating Watcher
<timestamp> [DEBUG] (cli) looping for data
<timestamp> [DEBUG] (watcher) starting watch
<timestamp> [DEBUG] (watcher) all pollers have started, waiting
for finish
<timestamp> [DEBUG] (redis) starting poll
<timestamp> [DEBUG] (service redis) querying Consul with &{...}
<timestamp> [DEBUG] (service redis) Consul returned 2 services
<timestamp> [DEBUG] (redis) writing data to channel
<timestamp> [DEBUG] (redis) starting poll
<timestamp> [INFO] (cli) received redis from Watcher
<timestamp> [INFO] (cli) invoking Runner
<timestamp> [DEBUG] (service redis) querying Consul with &{...}
# ...
```

FAQ

Q: How is this different than confd?

A: The answer is simple: Service Discovery as a first class citizen. You are also encouraged to read this Pull Request on the project for more background information. We think confd is a great project, but Consul Template fills a missing gap. Additionally, Consul Template has first class integration with Vault, making it easy to incorporate secret material like database credentials or API tokens into configuration files.

Q: How is this different than Puppet/Chef/Ansible/Salt?

A: Configuration management tools are designed to be used in unison with Consul Template. Instead of rendering a stale configuration file, use your configuration management software to render a dynamic template that will be populated by Consul.

Contributing

To build and install Consul Template locally, you will need a modern Go (Go 1.5+) environment.

First, clone the repo:

```
$ git clone https://github.com/hashicorp/consul-template.git
```

Next, download/update all the dependencies:

```
$ make updatedeps
```

To compile the consul-template binary and run the test suite:

```
$ make dev
```

This will compile the consul-template binary into bin/consul-template as well as your $GOPATH and run the test suite.

If you just want to run the tests:

```
$ make test
```

Or to run a specific test in the suite:

```
go test ./... -run SomeTestFunction_name
```

Submit Pull Requests and Issues to the Consul Template project on GitHub.